Arquitectura de Sistemes BigData pel Record Linkage de Xarxes Socials 2

Cristina Pérez Callejo

Resum– El desenvolupament de Xarxes, un projecte de recerca en humanitats digitals l'objectiu del qual és desenvolupar tècniques informàtiques per construir xarxes socials històriques, fa que el Centre de Visió per Computador (CVC) reculli una gran quantitat de dades que han de ser analitzades, tractades i emmagatzemades per la seva posterior utilització. El projecte que s'ha desenvolupat busca una solució per l'emmagatzematge de tota la Big Data que s'ha recopilat construint una arquitectura distribuïda, escalable horitzontalment, resistent a fallides i amb suport per diferents formats de dades. La solució aportada es basa en una estructura Hadoop multinode muntada sobre containers Docker. Aquesta estructura compta amb un servei Hive per gestionar els datasets emmagatzemats sota el HDFS i realitzar consultes, i un servei Sqoop per comunicar-se amb bases de dades alienes a l'estructura.
Paraules clau– hadoop, hive, sqoop, sistema distribuït, big data, mysql, derby, database, HDFS, docker, container

Abstract– The development of Xarxes, a research project in digital humanities whose aim is to develop computer tecniques to build social networks, makes the Computer Vision Center (CVC) gather a great amount of data that needs be analyzed, treated and stored for its later use. This project is in search of a solution for the storage of all the Big Data that has been compiled building a descentralized computing that is horizontally scalable , resistant to failure and wich work with different data formats. The solution provided is based on a multinode Hadoop structure built over Docker containers. This structure uses a Hive service to manage the datasets that are stored under the HDFS and to carry out enqueries, and also a Sqoop service to communicate with databases external to the structure.

Keywords– hadoop, hive, sqoop, distribuited system, big data, mysql, derby, database, HDFS, docker, container

1 Introducció

L'any 2015, el Centre de Visió per Computador (CVC) i el Centre d'Estudis Demogràfics (CED) de la Universitat Autònoma de Barcelona (UAB) comencen a treballar amb padrons històrics de la població, primer amb el projecte EINES i actualment amb XARXES, dos projectes de recerca en humanitats digitals, l'objectiu dels quals és desenvolupar tècniques informàtiques per a construir xarxes socials històriques.

Ambdós projectes compten amb una gran quantitat d'arxius que han de ser analitzats, tractats i emmagatzemats per la seva posterior utilització. El client, doncs, busca una estructura d'emmagatzematge de dades que es trobi en un sistema distribuït per tal de suportar tota aquesta Big Data.

Per aquest motiu es marquen dos objectius de desenvolupament:

- L'objectiu principal és la construcció d'una arquitectura d'emmagatzematge de dades per tal de poder processar tota la Big Data usada en el projecte Xarxes. L'estructura ha de ser distribuïda per tal de poder emmagatzemar la gran quantitat de dades en diferents nodes físics proporcionats pel client. Es busca que aquesta arquitectura sigui escalable horitzontalment per tal de poder ser ampliada en un futur per suportar noves insercions de dades. L'estructura, a més, ha de tenir certa capacitat de resiliència, ser resistent a fallides i caigudes i suportar dades en diferents formats, tant arxius com bases de dades. L'arquitectura podrà ser consultada per més d'un usuari alhora.

• E-mail de contacte: cristina.perezcal@gmail.com
• Menció realitzada: Enginyeria del Software
• Treball tutoritzat per: Oriol Ramos Terrades (Ciències Computació)
• Curs 2018/19

- L'objectiu secundari és la implementació d'una interfície d'usuari d'estil comandes per tal que els investigadors que tracten les dades puguin accedir a elles d'una forma fàcil i lògica. Com que actualment els investigadors usen Python per fer les seves operacions, es busca que la interfície pugui ser integrada en els seus usos actuals.

El document està estructurat de la següent forma: primer de tot es mostra la metodologia i la planificació seguida. A la següent secció s'especifiquen l'estat de l'art actual, el disseny implementat, el desenvolupament efectuat, el conjunt de resultats obtinguts i finalment les conclusions i les línies futures.

2 METODOLOGIA I PLANIFICACIÓ

Després d'avaluar els objectius del projecte es procedeix a la definició de la metodologia i la planificació.

2.1 Metodologia

La metodologia triada per desenvolupar el projecte ha estat una modificació de la metodologia AGILE, adaptant els seus principis a situació pròpia. Per una banda, el projecte es separa en dos blocs de treball formats per un conjunt d'sprints d'una setmana de duració.

Cada sprint produeix un entragable que ha sigut analitzat, dissenyat, desenvolupat i testejat, i que serà evolucionat al sprint següent. Els primers sprints es dediquen a una feina més teòrica i per tant són diferents a la resta.

2.2 Planificació

Per tal de poder complir els dos objectius plantejats es separa la planificació en dos grans blocs. Ambdós blocs comptem, però amb dues primera fase comunes:

Anàlisi: Es produeix una primera fase d'anàlisi i captació de requisits a partir de les reunions amb el client. A causa del caràcter agile de la metodologia, s'accepten petits canvis i ampliacions dels requisits durant tot el projecte, afegint els requisits que no poden ser implementats a l'apartat de línies futures.

Pressa de decisions i disseny general del sistema: es fa un primer disseny conceptual de l'arquitectura desenvolupada basant-nos en les especificacions del client. Per fer aquest disseny s'estudien les diverses eines actualment en el mercat que ens permetran muntar l'arquitectura desitjada. Com la secció anterior, el disseny efectuat queda obert a canvis.

Un cop realitzada les dues primeres fases es realitzen el conjunt de tasques que ens permetran assolir l'objectiu principal del treball. En aquest bloc, s'implementa el sistema de contenidors sobre el qual es desenvoluparà el treball i després es construeix l'estructura distribuïda. A continuació s'especifica el treball d'una forma més detallada.

Implementació dels contenidors: es realitza una formació bàsica sobre el sistema de contenidors (muntatge, manteniment, dependències...) i es busquen imatges del sistema distribuït per ser avaluades a la següent fase.

Implementació del sistema distribuït : es realitza una formació bàsica del sistema distribuït: com funciona, avantatges i inconvenients, eines amb què es relaciona... S'avalua les diferents configuracions del sistema distribuït (Únic node, node pseudodistribuït i multinode). S'avaluen les imatges seleccionades a la fase anterior. Es crea el sistema distribuït sobre els contenidors. Es realitzen proves per tal de buscar la millor distribució dels nodes. S'avaluen eines per tal de connectar el sistema creat amb bases de dades alienes a la configuració. Finalment es desenvolupa l'estructura triada i es fan proves per garantir que es compleixen els objectius de resiliència, resistència a caigudes, accessibilitat i suport de dades en diferents formats.

Pel segon objectiu, accés a les dades mitjançant una interfície gràfica de comandes, és necessari realitzar les següents tasques:

Implementació de Hive: es realitza una formació bàsica sobre l'eina Hive. S'avaluen les diferents implementacions referents al sistema de base de dades sobre el que es treballarà. S'implementa Hive sobre el node màster de l'arquitectura distribuïda i es fan proves de connexió. S'importen les dades. Es generen taules de tipus Hive. Es realitzen consultes. Es comprova que les taules formen part del HFSD. Es busca externalitzar el servei de Hive en un contenidor aliè.

Implementació del sistema d'accés a les dades: es dissenya la interfície de tipus comanda, es configura Hive per acceptar accés extern, es desenvolupa la interfície Python.

3 ESTAT DE L'ART

En aquesta secció es parlarà de la situació actual en el món de la Big Data. Es plantegen diverses eines que seran utilitzades pel desenvolupament del nostre projecte.

3.1 Contenidors

La virtualització del sistema operatiu és un mètode de virtualització del servidor en el qual el kernel del sistema operatiu permet l'existència de múltiples instàncies aïllades d'espai d'usuari. Aquesta virtualització, entre altres beneficis, redueix o elimina inconsistències entre les màquines de desenvolupament i les de producció, facilita treballar amb projectes amb diferents dependències i garanteix el desenvolupament en diferents sistemes operatius.

Hi ha dues grans companyies que treballen en el concepte d'aïllament de dues formes diferents, Docker [1] [2] [3] que fa servir contenidors i Vagrant [4], més semblant a una màquina virtual. Primer mostrarem característiques de cada una d'elles i finalment una comparativa entre diferents punts.

Docker no és una tecnologia en si mateixa sinó una forma d'accedir a ella. La finalitat de Docker, per tant és facilitar la creació i manipulació dels contenidors Linux, un tipus de màquina virtual lleugera. No reserva ni garanteix recursos a nivell Hardware i no disposa d'un aïllament total, pel que tot i que un cop aixecats els contenidors s'aïllen del sistema operatiu de la màquina host, en el moment de crear-se hi ha diferències significatives depenent del Sistema Operatiu del host.

Docker es compon de dos elements fonamentals: els contenidors Docker (un espai de treball que disposa dels elements necessaris perquè una aplicació funcioni) i les imatges Docker (un SO amb aplicacions instal·lades, l'element necessari per generar contenidors). Per tal d'automatitzar la pujada dels contenidors fa servir arxius yaml. YAML és un format de serialització de dades llegible per humans, basat en la identació i en el mapeig de les dades. Com a element a tenir en compte, cap a abril d'aquest any, Debian ha eliminat Wheezy i Jessie dels mirrors, fent que els arxius sources.list apuntin cap a repositoris que no contenen informació, el que fa que no es puguin executar apt-get updates des d'imatges antigues.

Vagrant és una eina que simplifica el flux de treball i redueix la càrrega de treball per executar i operar sobre màquines virtuals. Ens permet generar qualsevol entorn de desenvolupament basat en màquines virtuals i facilita els procediments per distribuir i compartir entorns de treball virtuals. Ofereix una interfície fàcil d'usar per crear servidors independents del sistema operatiu del desenvolupador de forma ràpida.

Un cop vista una primera definició d'ambdues eines, es produeix a fer una comparativa [5] entre elles. Comparant totes dues tecnologies, veiem que, per una banda, ens trobem que ambdues són sistemes multiplataforma, però mentre Docker només virtualitza sobre Linux, Vagrant ho fa també sobre Unix i Windows. Per altra banda, Vagrant reserva i garanteix els recursos a nivell Hardware i proporciona un aïllament total de l'estructura, mentre que Docker ho fa per software. Finalment, veiem que en l'àmbit temporal Docker és un sistema molt més ràpid de crear (minuts contra segons) i encara més ràpid d'arrencar.

3.2 Sistemes Gestor de Bases de Dades

Els Sistemes Gestors de Bases de Dades (SGBD) són un conjunt de programes que administren l'accés a una base de dades amb l'objectiu d'esdevenir una interfície entre aquesta, els usuaris i les aplicacions usades. Pel tipus de treball que farem servir, ens centrem en les bases de dades relacionals (SQL), més concretament en les següents:

- **Apache Derby:** és una base de dades open source implementada completament sobre Java. Com que es troba sota llicència Apache, és esdevé la metadata de Hive per defecte i està clarament integrada amb els serveis Apache. Derby ocupa molt poc espai, és molt fàcil d'instal·lar, deployar i usar, però tot i que suporta l'arquitectura client/servidor, no permet connexió per múltiples clients.
- **MySql[6]:** és el gestor de base de dades de codi obert més popular a la web. Compta amb compatibilitat amb SQL, una arquitectura client/sevidor, suport multiplataforma i unicode entre altres característiques. És usada com a metastore de Hive.
- **PostgreSQL:** és un gestor de bases de dades relacionades orientat a objectes que extèn funcions de SQL. Com en el cas de MySQL és disponible per multiplataformes i admet transaccions, subseleccions, disparadors i vistes entre altres característiques. És una altra alternativa per la metadata de Hive.

3.3 Sistemes de fitxers distribuïts

Els sistemes de fitxers distribuïts (DFS), són una forma d'organització d'arxius usant diversos servidor de fitxers sota un mateix espai de noms, proporcionant redundància, una millora del rendiment i tolerància a fallides. Actualment al mercat hi ha una competència entre Hadoop i Spark, dues eines que s'explicaran a continuació:

Hadoop és un sistema de codi obert que utilitzat per emmagatzemar, processar i analitzar grans volums de dades de diversos tipus: dades estructurades, no estructurades, semiestructurades, arxius de registres, imatges, vídeo... dins el seu sistema de fitxers distribuït, anomenat HDFS.

Aquesta eina es basa en l'arquitectura Master/slave on cada clúster Hadoop té un únic node màster i diversos nodes slave. Aquests nodes es troben dividits en racks, on cadascun dels rack pot comptar amb un màxim de 40 datanodes. Gràcies a la replicació de les dades en cada clúster es disposa d'una gran disponibilitat de les dades. Cada bloc d'arxius es replica en diversos datanodes en funció del factor de replicació del clúster. Un factor de replicació major de 1 implica que el bloc es troba localitzat en més d'un datanode dins del mateix rack. Entre els seus avantatges es troba que aïlla els desenvolupadors de dificultats presents en la programació paral·lela, permet distribuir un fitxer en diversos nòdes, és capaç d'executar processos en paral·lel i disposa de mòduls de control pel monitoratge de dades.

Hadoop compta amb eines per tal de tractar dades externes al seu HDFS. Apache Sqoop és una eina dissenyada per transferir dades entre bases de dades estructurades i Hadoop. Sqoop pot ser usat per importar dades des d'una base de dades externa cap al Sistema d'arxius distribuïts de Hadoop o cap a Hive o HBase. A més, pot ser usat per extreure les dades de Hadoop cap a external datastores.

Spark està basat en Hadoop Map Reduce i permet dividir o paral·lelitzar el treball si s'instal·la en diverses màquines. Està dissenyat per operar mitjançant el processament de fragments de dades en memòria, el que permet incrementar la velocitat dels processos, és idoni per aplicacions de Machine Learning. Spark no compta amb el seu propi sistema d'arxius pel que ha d'integrar-se amb altres per poder funcionar. Spark suporta el flux de dades acíclic, on cada tasca d'Spark genera un graf dirigit d'etapes de treball per un clúster determinat. Els clústers d'Spark es componen d'un programa que conté un SparkContext, aquest es comunica amb un conjunt de nodes treballadors mitjançant un Cluster Manager.

La comparativa [7] entre Hadoop i Spark es basa en tres conceptes:

- **Usabilitat**: en termes d'usabilitat, Spark disposa de diverses APIs que faciliten la seva usabilitat per part dels usuaris. Per altra banda, Hadoop disposa de complements com Pig i Hive que el fan més fàcil d'usar, tot i que no s'arriba a la simplicitat de les APIs.

- **Rendiment:** ambdues eines processen les dades de forma diferent, pel que no és trivial determinar quina aconsegueix el millor resultat. Per una banda, Spark treballa en memòria, fet que fa que els processos siguin més ràpids, però també que es necessiti més memòria per l'emmagatzemament. El seu rendiment pot resultar afectat amb l'ús d'aplicacions pesades. D'altra banda, Hadoop guarda les dades en el disc, pel que en termes generals és més lent de Spark. L'avantatge és que necessita un emmagatzematge inferior i, com elimina les dades que ja no són necessàries, no es produeixen pèrdues de rendiment amb apliacions pesades.

- **Seguretat:** en termes de seguretat, Hadoop proporciona a tots els usuaris els beneficis de Knox Gateway o Sentry. HDFS admet l'autorització en l'àmbit de servei, pel que es garanteix que els clients tenen els permisos adequats a nivell arxiu. Hadoop, a més, disposa de Hadoop YARN, una tècnica d'administració de clústers que desacobla les capacitats de gestió de recursos i planificació de MapReduce del component de processament de dades. Spark, d'altra banda, necessita executar-se en HDFS per accedir a permisos a nivell d'arxiu i tot i que suporta Hadoop YARN, el seu clúster natiu és Standalone.

3.4 Hive

Apache Hive [8] [9] és una infraestructura d'emmagatzematge de dades construïda sobre Hapoop que pot ser utilitzat tant per Apache Hadoop com per Apache Spark. Va ser creada per tal de poder fer possible l'anàlisi d'estructures SQL molt extenses i actualment és usada per empreses com Netflix i FINRA per tractar la seva BigData. Apache Hive suporta l'anàlisi de grans conjunts de dades emmagatzemats sota HDFS mitjançant un llenguatge de consultes basat en SQL anomenat HiveQL. Per defecte, Hive emmagatzema les seves metadades en una base de dades apache Derby, però es pot variar la seva configuració per fer-ho sobre MySQL o PosgreSQL per tal de permetre connexions múltiples.

4 Disseny

La següent secció explica el disseny de l'estructura implementada, explicant els diversos serveis trobats a cada un dels contenidors i la relació entre si.

Es decideix desenvolupar l'estructura sobre contenidors Docker perquè usen menys recursos, són més ràpids de crear i d'arrancar i més senzills de construir. El sistema distribuït triat per l'arquitectura és Hadoop per termes de seguretat i rendiment. Pel tipus de treball que es realitzarà, la velocitat del sistema no comporta un inconvenient.

El disseny implementat consta de set containers Dockers: un dataNode, quatre nodes slaves, una base de dades, un servidor d'administració de la base de dades, un servei de metastore Hive i un node Hive. A la Fig [1] es pot veure una representació de l'estructura.

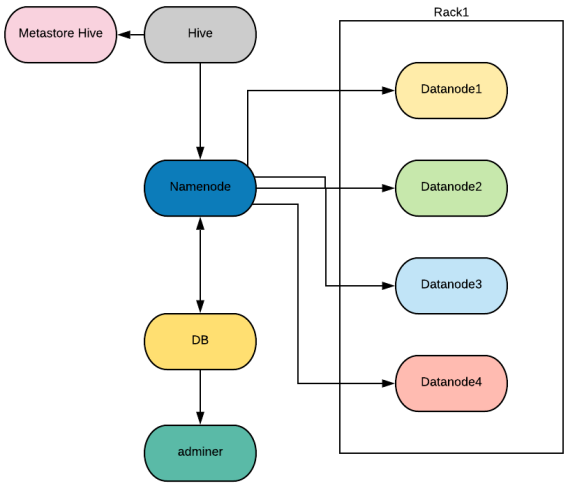

Fig. 1: Disseny de l'arquitectura

El conjunt de nodes es pot separar en dos grups, l'arquitectura distribuïda i una estructura que simula un entorn real. A continuació s'explica el paper [10] [11] de cadascun dels nodes.

- **Estructura distribuïda:** és l'estructura pròpia de l'arquitectura. Està formada per un únic rack i té un sistema de replicació de tres nodes i una mida de block de 128MB. Compta amb els nodes explicats a continuació.
  - **NameNode**: és l'encarregat d'administrar l'emmagatzematge HDFS, regulant l'accés per part dels clients als arxius guardats als datanodes i rastrejant i mantenint la localització els arxius de dades en HDFS guardant la informació d'on és cada bloc. Determina també la correlació de blocs en els datanodes i s'encarrega d'operacions d'apertura, tancat i renombrat d'arxius. Tota la seva informació s'emmagatzema en memòria el que permet una resposta ràpida. És, també, el repositori de les metadades de HDFS. Dins del contenidor es troba també el servei Sqoop per tractar la importació i exportació de dades entre HDFS i les diferents bases de dades. El sistema Sqoop base ha estat modificat amb diferents connectors per poder connectar-se a diferents tipus de bases de dades.
  - **Datanodes:** són els encarregats de guardar les dades del HDFS, mitjançant l'emmagatzemantze de blocs. Són els responsables de servir les sol·licituts de lectura i d'escriptura. Es comuniquen de forma periòdica amb el namenode indicant la llista de blocs que estan mantenint.
  - **Node Hive:** node amb el servei Hive. Està connectat amb el namenode per emmagatzemar les dades i amb el node Metastore per guardar la metadada.
  - **Node Metastore Hive:** és el repositori central remot de la metadata de Hive.

- **Estructura variable:** és el conjunt de Dockers que interactua amb l'arquitectura Hadoop. En el nostre disseny implementem una base de dades connectada a un servidor adminer per fer proves de connexió entre la base de dades i l'arquitectura HDFS. Aquesta estructura però, no forma part del sistema distribuït.

5 DESENVOLUPAMENT

A la següent secció es parlarà de les diferents fases de desenvolupament del projecte. Primer, es parlarà de la implementació del sistema distribuït, després de la implementació del servei Hive i finalment de la implementació de la interfície d'usuari.

5.1 Implementació del sistema distribuït

En aquesta secció es parlarà de la implementació del sistema distribuït Hadoop. Primer de tot, s'especifica la tria de la imatge Hadoop sobre la que treballarem, després el muntatge de Hadoop sobre els contenidors Docker i finalment la inserció de dades dins del sistema distribuït.

5.1.1 Tria de la imatge Hadoop

A causa de la falta d'una imatge oficial de Docker sobre Hadoop, es fa una comparativa entre diferents imatges usades per la comunitat per decidir la imatge base sobre la qual treballar. Aquesta comparativa es realitza muntant les diferents imatges i avaluant el contingut dels arxius de configuració (especialment en com es relacionen els nodes esclaus i el datanode, en la mida dels blocs i en el factor de replicació). Després d'aquesta comparativa veiem que no hi ha diferències significatives entre elles, però triem les imatges de Uhopper [12] com imatges base sobre les que es treballarà perquè són molt usades per la comunitat i els tipus de node que proporciona estan clarament diferenciats.

5.1.2 Muntatge de Hadoop sobre els contenidors Docker

En aquest apartat es mostra com s'implementa el sistema distribuït de Hadoop. Primer de tot, creem un arxiu docker-compose.yml (A.1) per tal d'afegir els serveis i automatitzar la creació i l'arrancada dels nodes. En aquest arxiu definim el nameNode i els datanodes sobre la mateixa xarxa i redirigim els ports que contenen serveis cap els seus homòlegs al sistema local per tal de poder fer crides fora del Docker.

Per tal de modificar els arxius interns de les imatges, vinculem els serveis amb un arxiu d'entorn, on especifiquem la direcció i nom del datanode, la mida del bloc i el factor de replicació.

Un cop construït, accedim al visualitzador de l'eina dins el port 50070 i comprovem que, tal com veiem a [Fig. 2] hem creat una estructura de quatre nodes actius.

5.1.3 Inserció de dades al HDFS

Un cop muntada l'estructura inserim el conjunt de dades sobre les quals es treballarà al HDFS des del datanode. Comprovem des del namenode que els fitxers existeixen, són accessibles i poden ser importats de nou fora de HDFS. A la [Fig. 3] podem veure el conjunt d'arxius que hem inserit.

Configured Capacity:	71.39 GB
DFS Used:	720 KB (0%)
Non DFS Used:	15.09 GB
DFS Remaining:	52.52 GB (73.58%)
Block Pool Used:	720 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	4 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	Sun Jun 16 16:55:14 +0200 2019
Last Checkpoint Time	Sun Jun 16 16:40:00 +0200 2019

Fig. 2: Arquitectura en funcionament

```
# bin/hadoop fs -ls /
  181305 2019-04-14 08:05 /Base_CastellviRosanes.xlsx
10667690 2019-04-14 08:07 /Base_links_SFLL_Definitiva.xlsx
       0 2019-04-14 08:04 /Datasets
   15915 2019-04-13 22:27 /NOTICE.txt
    1366 2019-04-13 22:24 /README.txt
   15915 2019-04-13 22:29 /home
       0 2019-04-11 21:29 /project
```

Fig. 3: Dades al HDFS

5.2 Connexió del sistema HDFS amb bases de dades externes

Per tal de simular un entorn real, generem una base de dades amb datasets aportats pel client. Per tal de generar aquesta base de dades, i degut a problemes que tenen els Docker de Windows amb els volums (per més informació llegir la secció 3.1), es decideix muntar la base de dades de forma híbrida, generant l'entorn de forma automàtica i l'estructura interna de forma manual.

5.2.1 Generació de la base de dades amb Docker-compose

Generem la base de dades a partir de la imatge oficial de mysql, afegint-hi el container als services del Docker-compose, i fent servir un altre container com a administrador per poder visualitzar els resultats en una interfície còmoda. Per tal d'evitar errors de connexió, és important que la base de dades i el visualitzador es trobin a la mateixa xarxa.

5.2.2 Importació de les dades dins la base de dades

Un cop automatitzada la creació del container, s'habilita de forma manual els volums compartits entre la màquina host (mitjançant l'eina Kitematic) i el Docker per tal de compartir els arxius necessaris per importar les dades. S'accedeix al sistema gestor de base de dades dins del container, es crea la base de dades, s'insereixen les taules i es comprova amb l'administrador que la base de dades està creada correctament.

Amb aquests passos hem simulat una un entorn real.

5.2.3 Connexió Base de dades amb Sqoop

Un cop la base de dades està muntada, es genera l'accés a serveis externs creant-hi un usuari i garantint-li accés total a les bases de dades amb les quals es comunicarà. Aquest usuari (que per la configuració de Sqoop ha de ser l'usuari root) és el que farem servir des del servei Sqoop del datanode per poder accedir a les dades.

Un cop la base de dades està configurada per acceptar la comunicació externa, modifiquem la imatge del datanode per tal d'afegir-li el servei Sqoop. Afegim els connectors amb bases de dades, que no venen per defecte amb el servei i hi instal·lem el jdk per poder fer servir el servei.

Amb el Sqoop està preparat, el connectem amb la base de dades mitjançant la direcció obtinguda en el mapeig de ports, en el nostre cas el 8083 i importem totes les taules localitzades a la base de dades cap a HDFS.

5.3 Implementació de HIVE

La següent secció explicarà com s'ha treballat amb l'eina Hive. A causa del caràcter agile de la implementació, es realitzen dues versions. La primera configuració es realitza instal·lant Hive sobre el datanode, la segona aïllant el servei en un docker separat. A la secció 6.5 es mostren els resultats de cada una de les configuracions. En aquesta secció, però, s'explica la configuració manual.

5.3.1 Configuració HIVE sobre node màster

Es decideix muntar Hive sobre el datanode de forma manual després d'adonar-nos del problema amb els aptget i per un problema de temps. Primer de tot, descarreguem HIVE a la màquina local per tal de poder fer diferents modificacions de forma persistent. Es modifica l'arxiu hive-config.sh per tal d'indicar el HADOOP-HOME on es realitzarà la lectura i escriptura de registres, i copiem la carpeta dins de namenode:/hive. Tot seguit, creem un directori dins de HDFS on es guardaran les taules creades amb HIVE i ens donem permisos. És important que aquesta carpeta sigui la definida als arxius de configuració. Amb aquests passos tenim configurat el servei Hive sobre el que treballarem.

5.3.2 Importació de dades a HIVE

Un cop tenim Hive configurat, fem les primeres proves d'importació de dades. En aquesta fase es necessita que els arxius d'entrada es trobin en format CSV, i que tots els datasets que es volen importar es trobin dins del sistema distribuït. Per tal de fer el conjunt de proves, insertem també un csv senzill amb un conjunt de dades número-lletra anomenat "test".

5.3.3 Creació de taules a HIVE

Un cop preparades totes les dades dins de Hadoop, entrem a Hive i creem primer la base de dades (BaseDeDadesTest) i després les taules que informarem amb les dades. Com a dada a tenir en compte, Hive és molt lent en iniciar-se. Tal com passava quan insertàvem els documents directament a Hadoop, la informació de la taula es replica en tres nodes. A la figura 4 podem veure que aquesta base de dades "test" no s'ha dividit en cap bloc (ja que la seva mida és petita) i s'ha replicat en els nodes 2, 3 i 1. A escala de HFDS, doncs, no hi ha diferencia entre aquest tipus de dades i les dades insertades sense control.

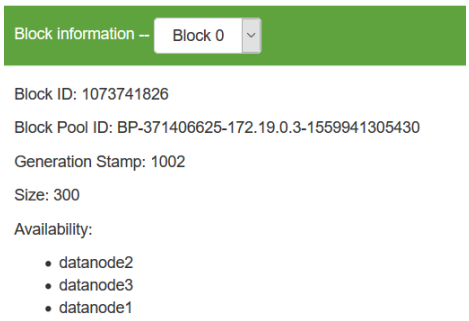

Fig. 4: Bloc creat per Hive

5.3.4 Consulta de dades a HIVE

Les consultes mitjançant Hive són molt semblants a les SQL, amb una sintaxi de l'estil de la següent: *Select * from Namestext limit 5;* seguit d'un codi (OK/FAIL) i en cas que sigui correcte es mostren les dades seleccionades amb la query. Després de fer diverses proves, veiem que el temps de càrrega és bastant estable sense importar quins nodes contenen la informació.

5.4 Implementació del sistema d'accés a les dades

En aquesta secció es realitza un disseny conceptual de com hauria de ser implementada la interfície de comandes demanada pel client i es fa un primer intent de connexió.

5.4.1 Disseny de la interfície de tipus comanda

Per tal de realitzar la interfície de comandes es planteja la utilització de l'eina PyHive per poder comunicar la nostra estructura amb un script python situat fora del sistema Docker. Per poder fer-la servir, l'eina necessita els mòduls sasl, thrift, thrift-sasl i PyHive.

Aquest script és un codi senzill basat en dues funcions, una primera funció anomenada **hiveconnection**, que rebrà com a paràmetres el hostName i el port del servei Hive, l'usuari, la contrasenya i la base de dades a la que es vol accedir. Aquesta funció retornarà el cursor necessari per fer les consultes.

La segona funció, **hiveconsult** rebrà com a paràmetres el cursor obtingut a la funció anterior, un string amb els camps que es volen obtenir, la taula sobre la qual es vol fer la consulta i el conjunt de condicions per limitar els resultats. La funció retornarà el valor de la consulta. A la [FIG. 5] es pot veure una possible primera versió del codi.

```
def hiveconnection(host_name, port, user,password, database):
    conn = hive.Connection(host=host_name, port=port, username=user,
                           password=password, database=database,
                           auth='CUSTOM')
    cur = conn.cursor()
    return cur;

def hiveconsult(cur, args, table, conditions=true){
    cur.execute('select '+args+' from '+table+' where '+conditions);
    result = cur.fetchall()
    return result
}
```

Fig. 5: Primera versió conceptual de la interfície d'usuari

5.4.2 Connexió HIVE des de Python

Per tal de ver visible el servei Hive fora del docker, exposem el port 10000 (port on segons la documentació es troba el servei), el mostrem fora de l'arquitectura i inicialitzem el hiveServer.

Intentem fer la connexió entre PyHive i Hive, però el sistema no és capaç de realitzar-la. Fem una mirada detallada dins del docker i no veiem cap error que pugui desencadenar aquesta fallida. Per falta de temps, el projecte es finalitza sense completar aquesta part.

6 RESULTATS

Com a resultat de finalitzar el projecte trobem que s'ha desenvolupat una arquitectura escalable, fàcil de modificar i ampliar, amb capacitat de resiliència, resistent a fallides i caigudes i amb capacitat de connexió simultània. A les subseccions 6.1 6.2, 6.3 i 6.5 es demostren aquestes característiques. A més, es mostren els resultats de variar la mida dels blocs de Hadoop i com aquest fet influeix en l'accessibilitat de les dades en cas de fallida del sistema.

6.1 Escalable horitzontalment

Per una banda, comprovem que tal com buscàvem, l'arquitectura desenvolupada és fàcilment escalable perquè podem afegir nous datanodes a l'arquitectura sense que afecti la resta.

Per tal de comprovar aquesta característica aixequem una estructura auxiliar de tres datanodes i hi afegim dades. Tot seguit reescrivim el Docker-compose per tal d'afegir-hi un quart datanode a l'arquitectura i hi tornem a inserir dades. Tal com es pot veure a la [Fig. 6], el sistema accepta el quart node i hi replica la informació en ell. La resta de blocs que es trobaven en els datanodes inicials segueix existint, pel que demostrem que el sistema és fàcilment escalable horitzontalment. D'aquesta forma es comprova també qué l'arquitectura darrera del namenode és completament transparent per l'usuari, que no percebrà l'ampliació del conjunt de datanodes.

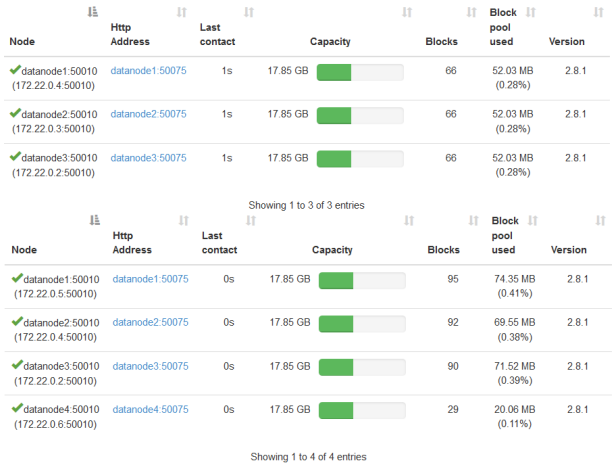

Fig. 6: Escalabilitat

6.2 Resiliència

Per tal d'avaluar la capacitat de resiliència de l'arquitectura, realitzem una simulació de caiguda dels diferents nodes fent servir el programa Kitematic per tal de parar i eliminar els containers. Les proves s'estructuren en tres fases: caiguda dels datanodes, caiguda del namenode i caiguda de l'estructura completa.

Per la primera prova, fem caure els diferents datanodes i els tornem a aixecar. Comprovem que els nodes tornen a connectar-se al namenode sense cap problema i el namenode pot accedir a la informació.

Per la segona prova, on comprovem la caiguda del namenode, el tombem i el tornem a aixecar mitjançant l'eina Kitematic. Es comprova que, com en el cas anterior, els datanodes es tornen a connectar i que es pot fer consultes.

Per la darrera prova de resiliència tombem tota l'estructura i la tornem a aixecar amb el docker-compose. Comprovem que tot i que s'ha perdut els fitxers auxiliars en el namenode, aquests se segueixen trobant a HDFS.

6.3 Resistent a fallides i caigudes

Per aquesta prova fem servir dues estructures diferents: una estructura de tres datanodes i un namenode (a la que a partir d'ara ens referirem com estructura A), i una de quatre datanodes i un namenode (estructura B). Totes dues estructures disposen d'una replicació de tres nodes i una mida de node de 128MB (major a tots els fitxers de prova).

En primer lloc, fem les diverses proves amb l'estructura A. Com tenim configurat que els blocs es tripliquin i es distribueixin, tots els datanodes contenen una còpia dels blocs. Per aquesta prova fem caure d'un en un els tres nodes i comprovem, tal com es pot veure a la [Taula 1], sempre que hi hagi com a mínim un dels nodes aixecat, es pot accedir a tota la informació.

TAULA 1: ACCÉS A LA INFORMACIÓ ESTRUCTURA A

Nodes caiguts	Accés a la informació
1	Si
2	Si
3	No

Per la segona prova, fem servir l'estructura B amb el mateix conjunt d'informació i obtenim els resultats mostrats a la [Taula 2]. Aquesta prova mostra certes similituds amb els resultats de la prova anterior: podem accedir a la informació si hi tenim un o dos nodes, però a diferència de l'estructura A, podem accedir a un conjunt d'informació si hi ha caigut un tercer node. Això és produeix a causa del fet que no tots els nodes contenen tota la informació.

TAULA 2: ACCÉS A LA INFORMACIÓ ESTRUCTURA B

Nodes caiguts	Accés a la informació
1	Si
2	Si
3	Depèn
4	No

Els resultats obtinguts en aquestes dues proves són fàcilment extrapolables a una estructura major. Amb un mateix factor de replicació, i per tant amb un mateix volum total de dades, obtenim una millor disponibilitat en sistemes amb més datanodes, si no busquem un fitxer en concret.

Aquesta disponibilitat, però, està clarament relacionada amb la mida dels blocs, tal com s'especifica a la següent subsecció.

6.4 Mida dels blocs

S'han realitzat diverses proves per tal d'avaluar la mida dels blocs que es replicaran a l'estructura. Aquestes proves es realitzen canviant el valor de mida de bloc en l'arxiu de env, no editant la imatge.

Primer de tot, muntem l'arquitectura master-slave amb 4 datanodes, indicant una mida de bloc de 1MB i hi afegim un arxiu de 10.17 MB. Tal com podem observar a la [FIG. 7], l'arxiu es divideix en 10 blocs que són repartits per l'arquitectura. En aquest cas, només un dels nodes conté tota informació sobre l'arxiu, mentre que els altres contenen informació parcial (entre 6 i 7 blocs).

En aquest cas, els resultats obtinguts en la secció anterior no funcionen. Això es produeix perquè es necessiten tots els blocs per tal de muntar la informació.

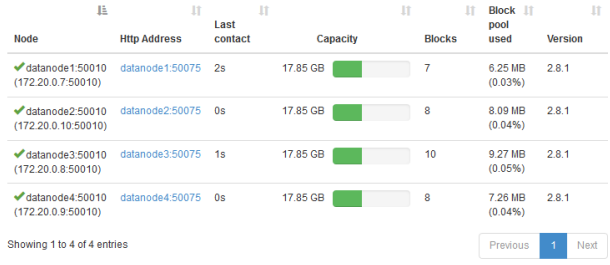

Fig. 7: Arxiu amb blocs 1MB.

Tornem a pujar la mateixa arquitectura indicant una mida de bloc de 128MB i tornem a pujar el mateix arxiu que en la prova anterior. En aquest cas, l'arxiu no se separa en cap bloc i tal com es pot veure a la [FIG. 8], es localitza totalment en tres nodes diferents. En aquest cas sí que es pot accedir a les dades tenint un node actiu.

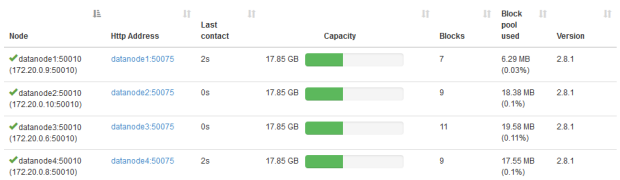

Fig. 8: Arxiu amb blocs 128MB

Amb aquesta prova comprovem que la mida del bloc és un factor decisiu per la resistència de fallides de l'estructura. Un valor petit és interessant per fer proves, però no és factible en construccions amb Big Data.

Com a resultat addicional, observem que l'estructura està preparada per acceptar diferents mides de blocs, tal com es mostra a la [Fig 9], fet que la fa altament flexible a canvis.

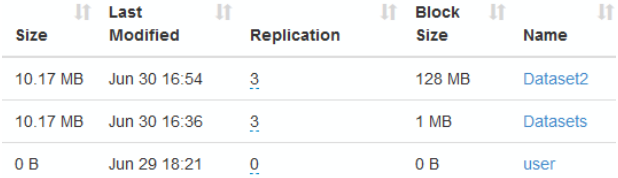

Fig. 9: Comparació mateix arxiu amb diferents blocs

6.5 Accessibilitat de les dades en paral·lel

Per tal de fer la següent prova muntem dos seveis Hive sobre el HDFS. El primer servei és un sistema Hive instal·lat amb metastore derby (Configuració A) i el segon un sistema Hive instal·lat amb metastore MySQL (Configuració B). La Configuració A ha estat muntada sobre el datanode i la Configuració B sobre un node aïllar degut a conflictes derivats dels problemes parlats a 3.1.

Per la Configuració A descarreguem HIVE al host i hi modifiquem els arxius de configuració per apuntar als entorns necessaris. Copiem el conjunt d'arxius al docker i hi comencem amb la configuració. Definim que el schema de hive sigui sobre derby. I quan fem les diferents proves veiem que no es pot accedir com a client a Hive mentre s'estan fent consultes.

Per la Configuració B fem servir una imatge Hive existent, que té com a restriccions la necessitat que el server MySQL per les metadades i el namenode tinguin un nom concret, pel que canviem al docker-compose els noms dels serveis. Inicialitzem el conjunt i veiem que l'schema s'inicialitza amb la composició del docker. Fem les mateixes proves que amb la Configuració A i veiem que amb aquesta configuració sí que podem accedir a Hive de forma simultània. Com a resultat, podem veure clarament que tot i la limitació de noms, és preferible una implementació aïllada i de tipus MySQL perquè permet connexió simultània de clients.

7 CONCLUSIONS

Com a conclusió del projecte, s'ha aconseguit crear una estructura distribuïda mitjançant l'eina Hadoop i s'afegeixen dades reals per tal de demostrar que és usable. Tal com havíem plantejat, aquesta arquitectura compleix amb les característiques de ser una arquitectura escalable, fàcil de modificar i ampliar, amb capacitat de resiliència, resistent a fallides i caigudes i amb capacitat de connexió simultània. L'objectiu principal, per tant, es compleix completament.

L'objectiu secundari, no obstant això, no ha estat finalitzat del tot. Per una banda s'ha aconseguit simplificar la dificultat de l'accés de les dades mitjançant l'eina Hive, però no s'ha finalitzat el desenvolupament de la interfície d'usuari a causa de problemes derivats d'una planificació poc

realista, problemes vinculats amb un canvi recent a les infraestructures Jessie i problemes per redirigir els ports dins del Docker de Hive.

Per tant, queda com a línia futura la finalització de la implementació de la interfície d'usuari.

AGRAÏMENTS

Per una banda m'agradaria agrair a la meva família per recolzar-me no només en el desenvolupament d'aquest treball, sinó durant tots aquests anys. M'agradaria agrair també a les meves amigues per suportar-me durant el transcurs del projecte, pels seus *"No sé de lo que me estás hablando, pero sigue explicando"* quan em poso a desvariar sobre coses tècniques i per posar el seu granet de sorra en aquest projecte.

REFERÈNCIES

[1] A. Gonzalez-Rodriguez. *DOCKER. Guía práctica*. RC Libros, 2017.

[2] JP. Gouigoux. *DOCKER. Primeros pasos y puesta en práctica de una arquitectura basada en micro servicios*. ENI, 2018.

[3] J. Turnbull. *The Docker Book*. Amazon Media EU, 2015.

[4] W. Gajda. *Pro Vagrant*. Apress, 2015.

[5] campusMVP. Docker vs vagrant: diferencias y similitudes y cuándo usar cada uno. url https://www.campusmvp.es/recursos/post/Docker-vs-Vagrant-diferencias-y-similitudes-y-cuando-usar-cada-uno.aspx, 2016. Accedit Maig-2019.

[6] F. Kromann. *Beginning PHP and MYSQL*. Apress, 2018.

[7] ————. Spark vs hadoop, ¿quién saldrá vencedor? url https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/spark-vs-hadoop-quien-saldra-vencedor, 2017. Accedit Maig-2019.

[8] Dayoung Du. *Apache Hive Essentials*. Packt Publishing, 2015.

[9] Capriolo D. Rutherglen J., Wampler D. *Programming Hive*. O'Reilly Media, 2012.

[10] R. D. Schneider. *Hadoop for Dummies*. John Wiley & Sons, 2012.

[11] IBM. Sistema de archivos distribuido de hadoop (hdfs). url https://www.ibm.com/support/knowledgecenter/es SPT3X_4.1.0/com.ibm.swg.im.infosphere.biginsights roduct.doc/doc/c0057606.html. Accedit Abril-2019.

[12] Uhooper. Docker. url Docker. https://bitbucket.org/uhopper/hadoop docker, 2016. Accedit Març-2019.

APÈNDIX

A.1 Docker compose

version: ”3”:
namenode:
image: uhopper/hadoop namenode:2.8.1
hostname: namenode
container_name: namenode
networks:
hadoop
volumes:
namenode:/hadoop /dfs/name
hive:/hive
env_file:
./hadoop basic.env
environment:
CLUSTER_NAME=hadoop
ports:
50070:50070
”10000:
datanode1:
image: uhopper/hadoop datanode:2.8.1
hostname: datanode1
contai ner_name: datanode1
networks:
hadoop
volumes:
datanode1:/hadoop/dfs/data
env_file:
./hadoop basic.env
expose:
8020
datanode2:
image: uhopper/hadoop datanode:2.8.1
hostname: datanode2
container_n ame: datanode2
networks:
hadoop
volumes:
datanode2:/hadoop/dfs/data
env_file:
./hadoop basic.env
expose:
8020
datanode3:
image: uhopper/hadoop datanode:2.8.1
hostname: datanode3
container_name: datanode3
networks:
hadoop
volumes:
datanode3:/hadoop/dfs/data
env_file:
./hadoop basic.env
expose:
8020
datanode4:
image: uhopper/hadoop datanode:2.8.1
hostname: datanode4
container_name: datan ode4
networks:
hadoop
volumes:
datanode4:/hadoop/dfs/data
env_file:
./hadoop basic.env
expose:
8020
networks:
hadoop:
volumes:
namenode:
datanode1:
datanode2:
datanode3:
datanode4:
hive:

A.2 ENV

CORE_CONF_fs_defaultFS= hdfs://hadoop-master:8020
Configure default BlockSize and Replication for local data.
HDFS_CONF_dfs_blocksize= 128m

YARN_CONF_yarn_log___aggregation___enable= true
YARN_CONF_yarn_resourcemanager_recovery _enabled=
true
YARN_CONF_yarn_resourcemanager_store_class=
org.apache.hadoop.yarn.server.resourcemanager.recovery
.FileSystemRMStateStore
YARN_CONF_yarn_resourcemanager_fs_state___store_uri=
/rmstate
YARN_CONF_yarn_nodemanager_remote___app___log___dir=
/app-logs
YARN_CONF_yarn_log_server_url=
http://historyserver:8188/applicationhistory/logs/
YARN_CONF_yarn_timeline___service_enabled= true
YARN_CONF_yarn_timeline___service_generic___application___history
_enabled=true
YARN_CONF_yarn_resourcemanager_system___metrics___publisher
_enabled=true
YARN_CONF_yarn_resourcemanager_hostname= re-
sourcemanager
YARN_CONF_yarn_timeline___service_hostname= history-
server

A.3 Diagrama de gantt

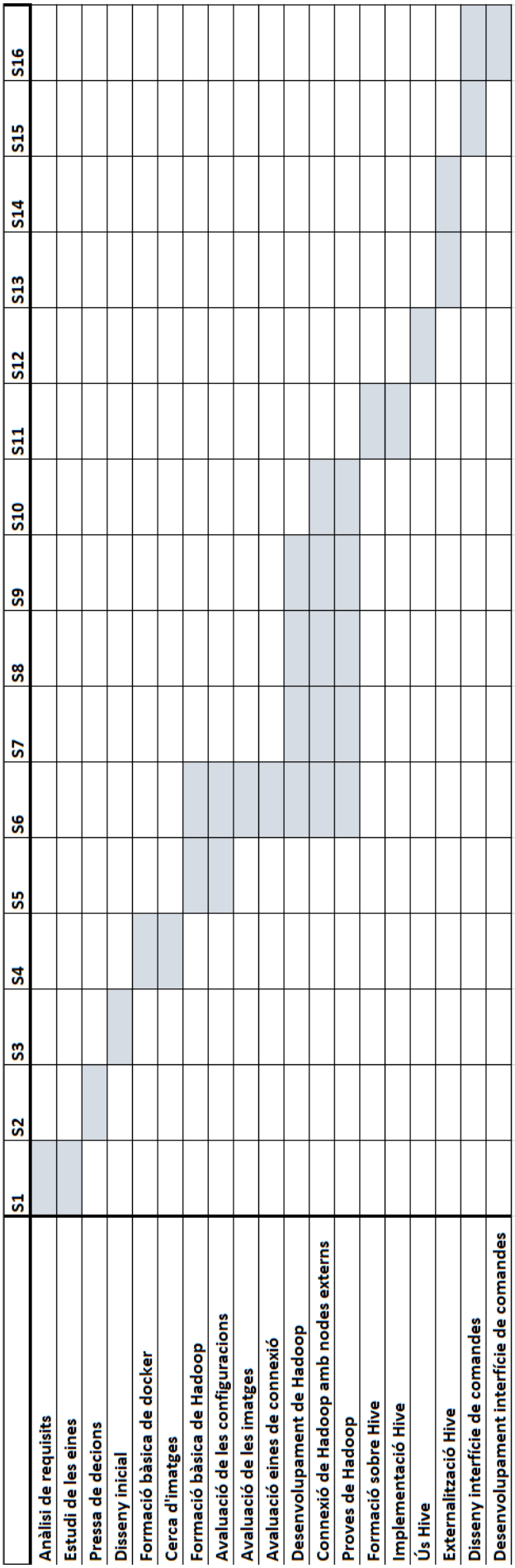

Fig. 10: Diagrama de Gantt